ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH, INDORE

ACROPOLIS
Enlightening Wisdom

Lab Manual of Machine Learning [CSIT-602]

B. Tech. VI Semester

Jan -June 2023

Department of Computer Science and Information Technology

Submitted to

Pro. Nidhi Nigam

Submitted By

Ashish Maley
0827CI201042

ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH,INDORE

Department of Computer Science and Information Technology

<u>Certificate</u>

This is to certify that the experimental work entered in this journal as per the B Tech III year syllabus prescribed by the RGPV was done by Mr. Ashish Maley B.Tech VI semester CI in the Machine Learning Laboratory of this institute during the academic year Jan June 2023

Signature of Faculty

INDEX PAGE

Programs to be uploaded on Github
Github Link:

Experiment No	Program	Commit date (in Github)	Sign of faculty
1	Python Basic Programming including Python Data Structures such as List, Tuple, Strings, Dictionary, Lambda Functions, Python Classes and Objects and Python Libraries such as Numpy, Pandas, Mat plotlib etc.		
2	Python List Comprehension with examples		
3	Basic of Numpy, Pandas and Matplotlib		
4	Brief Study of Machine Learning Frameworks such as Open CV, Scikit Learn, Keras, Tensorflow etc.		
5	For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation of **Linear Regression Algorithm**		
6	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Implementation of **Linear Regression Algorithm** Linear Regression using Python library (for any given CSV dataset)		
7	For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation for binary classification using **Logistic Regression Algorithm**		
8	Build an **Artificial Neural Network (ANN)** by implementing the Backpropagation algorithm and test the same using MNIST **Handwritten Digit Multiclass classification** data sets with use of use of batch normalization, early stopping and drop out		
9	Build an **Artificial Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100**		

	Multiclass classification data sets with use of use of batch normalization, early stopping and drop out		
10	ANN implementation use of batch normalization, early stopping and drop out (For Image Dataset such as Covid Dataset)		
11	Build an **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **MNIST Handwritten Digit Multiclass classification** data sets.		
12	Build an **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100 Multiclass classification** data sets.		
13	Implementation of Transfer Learning (VGG 16)		
14	Implementation of RNN		

EXPERIMENT 1

AIM: Python Basic Programming including Python Data Structures such as List, Tuple, Strings, Dictionary, Lambda Functions, Python Classes and Objects and Python Libraries such as Numpy, Pandas, Mat plotlib etc. PROGRAM/ IPYNB FILE:

*****List*****

```
List = [22, 21, 3, 2.3]
print("\n",List)
```

*****Tuple*****

```
Tuple = ('Ashish', 'For')
print("\nTuple with the use of String: ") print(Tuple)
```

*****String*****

```
String = "Welcome to AITR" print("\nCreating
String: ") print(String)
print("\nFirst character of String is: ",String[0]) print("\nLast character of String is:
",String[-1])
```

*****Dictionary*****

```
Dict = {'Ashish': 'Maley', 'List' : [1, 2, 3, 4]}

print("\nCreating Dictionary: ")

print(Dict)

print("\nAccessing an element using key: Ashish")

print(Dict['Ashish'])

print("\nAccessing an element using get: List")

print(Dict.get("List"))



myDict = {x: x**2 for x in [1,2,3,4,5]}

print(myDict)

List = [1, 2, 3, 2.3]

print(List)
```

Lambda Function

```
str1 = 'Good Morning'
```

```
rev_upper = lambda string: string.upper()[::-1]
print(rev_upper(str1))
```

Python Classes and Object class Bike:

```
        name = "" gear = 0
bike1 = Bike() bike1.gear = 11
bike1.name = "Mountain Bike"
print(f"Name: {bike1.name}, Gears: {bike1.gear} ")
```

Numpy

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5]) print(arr)
print(type(arr))
```

Panda

```
import pandas as pd
df = pd.read_csv('data.csv') print(df.to_string())
```

Mathplot Lib import sys

```
import matplotlib matplotlib.use('Agg')
import matplotlib.pyplot as plt import numpy
as np
xpoints = np.array([0, 6]) ypoints =
np.array([0, 250]) plt.plot(xpoints, ypoints)
plt.show() plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

RESULTS:

List

```
List = [22, 21, 3, 2.3]
print("\n",List)
```

```
 [22, 21, 3, 2.3]
```

Tuple

```
Tuple = ('Ashish', 'For')
print("\nTuple with the use of String: ")
print(Tuple)
```

String

```
String = "Welcome to AITR"
print("\nCreating String: ")
print(String)
print("\nFirst character of String is: ",String[0])
print("\nLast character of String is: ",String[-1])
```

```
Creating String:
Welcome to AITR

First character of String is:  W

Last character of String is:  R
```

Dictionary

```
Dict = {'Ashish': 'Maley', 'List' : [1, 2, 3, 4]}
print("\nCreating Dictionary: ")
print(Dict)
print("\nAccessing an element using key: Ashish")
print(Dict['Ashish'])
print("\nAccessing an element using get: List")
print(Dict.get("List"))

myDict = {x: x**2 for x in [1,2,3,4,5]}
print(myDict)
List = [1, 2, 3, 2.3]
print(List)
```

Lambda Function

```
str1 = 'Good Morning'
rev_upper = lambda string: string.upper()[::-1]
print(rev_upper(str1))
```

```
GNINROM DOOG
```

Python classes and object

```
class Bike:
  name = ""
  gear = 0
bike1 = Bike()
bike1.gear = 11
bike1.name = "Mountain Bike"
print(f"Name: {bike1.name}, Gears: {bike1.gear} ")
```

```
Name: Mountain Bike, Gears: 11
```

Numpy

```python
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

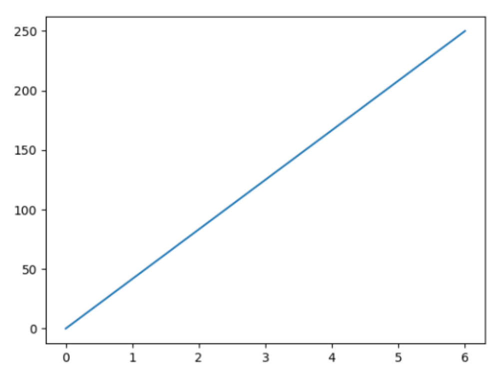
```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Panda

```python
import pandas as pd
df = pd.read_csv('Data.csv')
print(df.to_string())
```

```
   S.No.  Country   Age   Salary Purchased  Unnamed: 5
0      1   France  44.0  72000.0        No         NaN
1      2    Spain  27.0  48000.0       Yes         NaN
2      3  Germany  30.0  54000.0        No         NaN
3      4    Spain  38.0  61000.0        No         NaN
4      5  Germany  40.0      NaN       Yes         NaN
5      6   France  35.0  58000.0       Yes         NaN
6      7    Spain   NaN  52000.0        No         NaN
7      8   France  48.0  79000.0       Yes         NaN
8      9  Germany  50.0  83000.0        No         NaN
9     10   France  37.0  67000.0       Yes         NaN
```

Matplotlib



EXPERIMENT 2

Aim : Python List Comprehension with examples

```
[ ] x = [i for i in range(15)]
    x

    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
a = [4,6,7,3,2]
b = [x*2 for x in a if x > 5]
```

```
[ ] my_list = [2, 5, 8, 11, 15]
    new_list = [x*2 if x < 10 else x+1 for x in my_list]
    print(new_list)

    [4, 10, 16, 12, 16]
```

```
[ ] my_list = [i for i in range(1, 30) if i%2==0 if i%5==0]
    print(my_list)
```

```
[ ] names = ['Ch','Dh','Eh','cb','Tb','Td']
    new_names = [name for name in names if name.lower().startswith('c')]
    new_names
```

```
[ ] new_list = [num * 2 for num in range(5)]
    new_list
```

```
[ ] new_list = [num for num in range(50) if  num > 20 and num % 2 == 0]
    new_list
```

```
[ ] fave_language_chars = [letter for letter in "Python"]
    fave_language_chars
```

```
#Count the number of spaces in a string
sentence = 'the slow solid squid swam sumptuously through the slimy swamp'
my_list = [i for i in sentence if i==' ']
print(len(my_list))
```

```
# Create a list of all the consonants in the string "Yellow Yaks like yelling and yawning and yesturday they yodled while eating yuky yam
sentence = "Yellow Yaks like yelling and yawning and yesturday they yodled while eating yuky yams"
my_list = [i for i in sentence if i not in 'a,e,i,o,u, ']
my_list
```

```
# Get the index and the value as a tuple for items in the list ["hi", 4, 8.99, 'apple', ('t,b','n')].  Result would look like [(index, va
v =[ 'a','e','i','o','u',' ' ]
r = [(i,j) for i,j in enumerate(v)]
r
```

```
# Find the common numbers in two lists (without using a tuple or set) list_a = [1, 2, 3, 4], list_b = [2, 3, 4, 5]
list_a = [1, 2, 3, 4]
list_b = [2, 3, 4, 5]
com = [i for i in list_a if i in list_b]
com

    [2, 3, 4]
```

```
# Get only the numbers in a sentence like 'In 1984 there were 13 instances9 of a protest with over 1000 people attending'.  Result is a l
sentence = 'In 1984 there were 13 instances of a protest with over 1000 people attending'
word = sentence.split( )
com = [i for i in word if not i.isalpha()]
com

    ['1984', '13', '1000']
```

EXPERIMENT 3

Aim: Basic of Numpy, Pandas and Matplotlib

NumPy

NumPy is a Python library for numerical computing. It provides powerful tools for working with arrays and matrices.

Creating Arrays

Here's how to create a 1D array and a 2D array using NumPy:

```
import numpy as np

# Create a 1D array
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)

# Create a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2)
```

Basic Operations

NumPy provides many functions for performing basic operations on arrays. Here are a few examples:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Element-wise addition
result = arr1 + arr2
print(result)

# Element-wise multiplication
result = arr1 * arr2
print(result)

# Dot product
result = np.dot(arr1, arr2)
print(result)
```

Pandas

Pandas is a Python library for data manipulation and analysis. It provides powerful tools for working with tabular data.

Reading Data

Here's how to read a CSV file using Pandas:

```
import pandas as pd
```

```
# Read a CSV file
data = pd.read_csv("data.csv")

# Print the first 5 rows
print(data.head())
```

Selecting Data

Pandas provides many ways to select and filter data. Here are a few examples:

```
import pandas as pd

# Read a CSV file
data = pd.read_csv("data.csv")

# Select a single column
column = data["column_name"]
print(column)

# Select multiple columns
columns = data[["column1", "column2"]]
print(columns)

# Filter rows based on a condition
filtered_data = data[data["column_name"] > 5]
print(filtered_data)
```

Matplotlib

Matplotlib is a Python library for creating visualizations. It provides powerful tools for creating charts and graphs

```
import matplotlib.pyplot as plt
import numpy as np

# Generate some data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create a line plot
plt.plot(x, y)

# Add labels and a title
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('A Simple Plot')

# Show the plot
plt.show()
```

EXPERIMENT 4

Aim : Brief Study of Machine Learning Frameworks such as Open CV, Scikit Learn, Keras, Tensorflow etc.

OpenCV
OpenCV (Open Source Computer Vision) is a library of computer vision and machine learning algorithms. It provides tools for image and video analysis, object detection, face recognition, and more.
OpenCV is written in C++ but provides Python bindings for easy integration with Python applications. It is widely used in industries like robotics, self-driving cars, and security.
Scikit-learn
Scikit-learn is a Python library for machine learning. It provides tools for data preprocessing, model selection, evaluation, and visualization.
Scikit-learn includes a variety of algorithms for classification, regression, clustering, and dimensionality reduction. It also provides tools for feature extraction and selection.
Scikit-learn is built on top of NumPy, SciPy, and Matplotlib, and is widely used in academic research and industry.
Keras
Keras is a high-level neural network library written in Python. It provides a simple interface for building deep learning models.
Keras supports a wide range of neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and generative adversarial networks (GANs).
Keras can run on top of other deep learning frameworks like TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK).
TensorFlow
TensorFlow is a powerful open-source deep learning framework developed by Google. It provides tools for building and training deep neural networks.
TensorFlow supports a wide range of neural network architectures and provides a variety of optimization algorithms and regularization techniques.
TensorFlow can run on CPUs, GPUs, and even distributed systems, making it a powerful tool for training large-scale deep learning models.
Overall, these four machine learning frameworks are powerful tools for building and training machine learning models. Each has its own strengths and weaknesses, so it's important to choose the right tool for the job at hand.

EXPERIMENT 5

AIM : For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation of Linear Regression Algorithm

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]

```
#Step 1: Data Preprocessing
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('/content/studentscores.csv')
X = dataset.iloc[ : ,   : 1 ].values
Y = dataset.iloc[ : , 1 ].values

X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 1/4, random_state = 0)
```

```
X_train
```

```
array([[7.8],
       [6.9],
       [1.1],
       [5.1],
       [7.7],
       [3.3],
       [8.3],
       [9.2],
       [6.1],
       [3.5],
       [2.7],
       [5.5],
       [2.7],
       [8.3],
       [9.2],
       [6.1],
       [3.5],
       [2.7],
       [5.5],
       [2.7],
       [8.5],
       [2.5],
       [4.8],
       [8.9],
       [4.5]])
```

```
Y_train
```

```
array([86, 76, 17, 47, 85, 42, 81, 88, 67, 30, 25, 60, 30, 75, 21, 54, 95,
       41])
```

Step 2: Fitting Simple Linear Regression Model to the training set

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor = regressor.fit(X_train, Y_train)
```

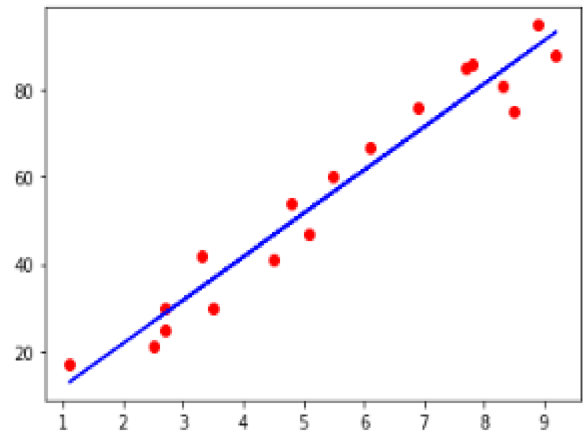
Step 3: Predecting the Result

```
Y_pred = regressor.predict(X_test)
```

Step 4: Visualization

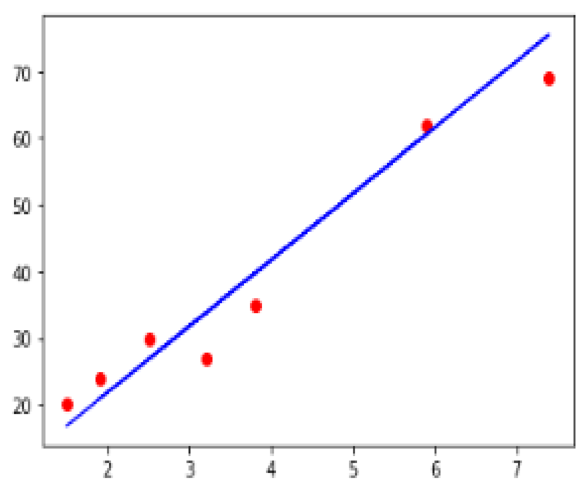
```
[ ] plt.scatter(X_train , Y_train, color = 'red')
    plt.plot(X_train , regressor.predict(X_train), color ='blue')
```

```
[<matplotlib.lines.Line2D at 0x7f52e99adb80>]
```




```
[ ] plt.scatter(X_test , Y_test, color = 'red')
    plt.plot(X_test , regressor.predict(X_test), color ='blue')
```

```
[<matplotlib.lines.Line2D at 0x7f52e99a5100>]
```



EXPERIMENT 6

AIM : For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation for binary classification using Logistic Regression Algorithm

```
[40] import csv
```

```
[41] import numpy as np
```

```
[42] import matplotlib.pyplot as plt
```

```
[43] import pandas as pd
```

```
[44] def normalize(X):

         #function to normalize feature matrix, X

         mins = np.min(X, axis = 0)
         maxs = np.max(X, axis = 0)
         rng = maxs - mins
         norm_X = 1 - ((maxs - X)/rng)
         return norm_X
```

```
[45] from sklearn import datasets, linear_model, metrics
     import csv
     import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
     dataset = pd.read_csv('dataset1 (2).csv')
```

```
[45] from sklearn import datasets, linear_model, metrics
     import csv
     import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
     dataset = pd.read_csv('dataset1 (2).csv')
     dataset=np.array(dataset)
     X = normalize(dataset[:, :-1])
```

stacking columns wth all ones in feature matrix

```
[46] X = np.hstack((np.matrix(np.ones(X.shape[0])).T, X))
     print('\n')
     print(X)
```

```
 [1.         0.50157705 0.20533614]
 [1.         0.71608093 0.36000054]
 [1.         0.84542406 0.45600519]
 [1.         0.88958149 0.62399643]
 [1.         1.         0.7280026 ]
 [1.         0.86750277 0.55199632]
 [1.         0.70977273 0.54933366]
 [1.         0.64037001 0.48533506]
 [1.         0.22713276 0.46133052]
 [1.         0.12934312 0.335996  ]
 [1.         0.09778964 0.2453302 ]
 [1.         0.42586613 0.44000216]
 [1.         0.31546009 0.34933636]
 [1.         0.57097977 0.55467251]
 [1.         0.40063331 0.49599924]
 [1.         0.2870607  0.44000216]
 [1.         0.40063331 0.57066202]
 [1.         0.52365577 0.664004  ]
 [1.         0.49842295 0.59466656]
```

```
[47] y = dataset[:, -1]
```

splitting X and y into training and testing sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)
```

create logistic regression object

```
[49] reg = linear_model.LogisticRegression()
```

train the model using the training sets

```
[50] # reg.fit(X_train, y_train)
```

```
[51] #1D Array
     import numpy as np


     arr = np.array([1, 2, 3, 4, 5])


     print(arr)
     print(arr.shape)
```

```
[1 2 3 4 5]
(5,)
```

```
[52] # 2 D Array (Matrix)
     import numpy as np


     arr = np.array([[1, 2, 3], [4, 5, 6]])


     print(arr)
     print(arr.shape)
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
```

```
[53] import numpy as np


     arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
     print(arr)
     print(arr.shape)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
(2, 2, 3)
```

```
[54] import numpy as np


     arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
     print(arr)
     print(arr.shape)
     print(np.ones(3))
```

```
    [[1 2 3]
54]   [4 5 6]]

     [[1 2 3]
      [4 5 6]]]
    (3, 2, 3)
    [1. 1. 1.]
    [[1. 1.]
     [1. 1.]
     [1. 1.]]
    [0. 0. 0.]
    [[0. 0. 0. 0.]
     [0. 0. 0. 0.]
     [0. 0. 0. 0.]]
```

```
55] import csv
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

```
56] def normalize(X):

        #function to normalize feature matrix, X

        mins = np.min(X, axis = 0)
        maxs = np.max(X, axis = 0)
        rng = maxs - mins
        norm_X = 1 - ((maxs - X)/rng)
        return norm_X
```

```
57] def logistic_func(beta, X):


        # below is the code for 1/1+e^(-(bo*x1+ b1*x2 +  b1*x3............ ))
        #return  horizontal array


        return 1.0/(1 + np.exp(-np.dot(X, beta.T)))
```

Logistic_Regression 44.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[56] def normalize(X):

        #function to normalize feature matrix, X

        mins = np.min(X, axis = 0)
        maxs = np.max(X, axis = 0)
        rng = maxs - mins
        norm_X = 1 - ((maxs - X)/rng)
        return norm_X
```

```
[57] def logistic_func(beta, X):


        # below is the code for 1/1+e^(-(bo*x1+ b1*x2 +  b1*x3........... ))
        #return  horizontal array


        return 1.0/(1 + np.exp(-np.dot(X, beta.T)))
```

```
[58] def log_gradient(beta, X, y):


        #first_calc = y_prediction - y_actual  for all samples
        first_calc = logistic_func(beta, X) - y.reshape(X.shape[0], -1)


        # now in below step we will find the partial derivative
        #final_calc= gradient is (y_prediction - y_actual)*x  for all samples


        final_calc = np.dot(first_calc.T, X)

        return final_calc
```

```
[59] def cost_func(beta, X, y):
```

✓ 0s completed at 12:00 AM

Logistic_Regression 44.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
[56] def normalize(X):

         #function to normalize feature matrix, X

         mins = np.min(X, axis = 0)
         maxs = np.max(X, axis = 0)
         rng = maxs - mins
         norm_X = 1 - ((maxs - X)/rng)
         return norm_X
```

```
[57] def logistic_func(beta, X):


         # below is the code for 1/1+e^(-(bo*x1+ b1*x2 +  b1*x3........... ))
         #return  horizontal array


         return 1.0/(1 + np.exp(-np.dot(X, beta.T)))
```

```
[58] def log_gradient(beta, X, y):


         #first_calc = y_prediction - y_actual  for all samples
         first_calc = logistic_func(beta, X) - y.reshape(X.shape[0], -1)


         # now in below step we will find the partial derivative
         #final_calc= gradient is (y_prediction - y_actual)*x  for all samples


         final_calc = np.dot(first_calc.T, X)

         return final_calc
```

```
[59] def cost_func(beta, X, y):
```

0s complete

Logistic_Regression 44.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
[56] def normalize(X):

         #function to normalize feature matrix, X

         mins = np.min(X, axis = 0)
         maxs = np.max(X, axis = 0)
         rng = maxs - mins
         norm_X = 1 - ((maxs - X)/rng)
         return norm_X
```

```
[57] def logistic_func(beta, X):

         # below is the code for 1/1+e^(-(bo*x1+ b1*x2 +  b1*x3........... ))
         #return  horizontal array

         return 1.0/(1 + np.exp(-np.dot(X, beta.T)))
```

```
[58] def log_gradient(beta, X, y):

         #first_calc = y_prediction - y_actual  for all samples
         first_calc = logistic_func(beta, X) - y.reshape(X.shape[0], -1)

         # now in below step we will find the partial derivative
         #final_calc= gradient is (y_prediction - y_actual)*x  for all samples

         final_calc = np.dot(first_calc.T, X)

         return final_calc
```

```
[59] def cost_func(beta, X, y):
```

0s complete

Logistic_Regression 44.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
[62]     print('\n')
         print(beta)


         # beta values after running gradient descent
         beta, num_iter, cost_per_iter = train(X, y, beta)
         itr = np.arange(1,num_iter+1)
         print("\n")
         print("Cost_per_iteration = ", np.array(cost_per_iter).T)
         #plt.plot(cost_per_iter, itr)


         # estimated beta values and number of iterations
         print('\n')
         print("Estimated regression coefficients:", beta)
         print("No. of iterations:", num_iter)
         # predicted labels
         y_pred = pred_values(beta, X)

         # number of correctly predicted labels
         print("\n")
         print("Correctly predicted labels:", np.sum(y == y_pred))

         # plotting regression line
         plot_reg(X, y, beta)
```

```
[[0.19242517 0.05065823]
 [0.41640382 0.09866732]
 [0.6119831  0.17599276]
 [0.69084812 0.29600195]
 [0.6119831  0.27999892]
 [0.68453991 0.35733787]
 [0.74448032 0.44266483]
 [0.73501801 0.33867218]
 [0.90536447 0.4826724 ]
 [0.81071647 0.36800205]
 [0.83911585 0.4640067 ]
 [0.85488637 0.61599492]
 [0.70661863 0.51200227]
 [0.55520926 0.3813289 ]
```

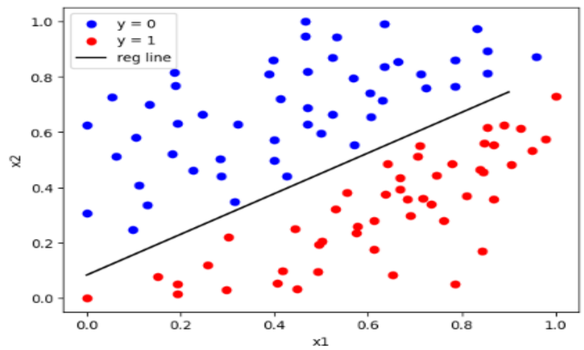
0s

Logistic_Regression 44.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[1.         0.46686946 0.94593572]]

[[0. 0. 0.]]
Cost is: 68.62157087543457

Cost_per_iteration =  [6.86215709e+01 1.07779650e+00 1.02619087e+00 ... 1.00111937e-03
 1.00051155e-03 9.99904342e-04]

Estimated regression coefficients: [[  1.6908351   15.00842207 -20.4138904 ]]
No. of iterations: 2613

Correctly predicted labels: 99
```



Let's Try To Plot a Sigmoid

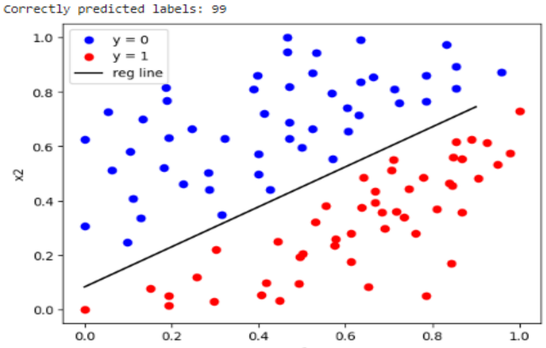
✓ 0s

Logistic_Regression 44.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[1.         0.46686946 0.94593572]]

[[0. 0. 0.]]
Cost is: 68.62157087543457

Cost_per_iteration =  [6.86215709e+01 1.07779650e+00 1.02619087e+00 ... 1.00111937e-03
 1.00051155e-03 9.99904342e-04]

Estimated regression coefficients: [[  1.6908351   15.00842207 -20.4138904 ]]
No. of iterations: 2613

Correctly predicted labels: 99
```

Let's Try To Plot a Sigmoid

✓ 0s completed

EXPERIMENT 7

AIM : Build an Artificial Neural Network (ANN) by implementing the Backpropagation algorithm and test the same using MNIST Handwritten Digit Multiclass classification data sets with use of use of batch normalization, early stopping and drop out

ANN lab work.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 28

+ Code + Text

Loading the MNIST dataset in Keras

```
[ ] from keras.datasets import mnist
    (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

The images are encoded as Numpy arrays, and the labels are an array of digits, ranging from 0 to 9.

```
[ ] train_images.shape
    len(train_labels)
    train_labels
    test_images.shape
    len(test_labels)
    test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

The network architecture

```
[ ] from keras import models
    from keras import layers
    network = models.Sequential()
    # Dense(32) is a fully-connected layer with 32 hidden units.
    # in the first layer, you must specify the expected input data shape :
    # here, 28 X 28=784 -dimensional vectors.
    network.add(layers.Dense(32, activation='sigmoid', input_shape=(28 * 28, )))
    network.add(layers.Dense(8, activation='sigmoid'))
    network.add(layers.Dense(10, activation='softmax'))
```

ANN lab work.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 28

+ Code + Text

```
[ ] Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     dense (Dense)               (None, 32)                25120

     dense_1 (Dense)             (None, 8)                 264

     dense_2 (Dense)             (None, 10)                90

    =================================================================
    Total params: 25,474
    Trainable params: 25,474
    Non-trainable params: 0
    _________________________________________________________________
    Mounted at /content/drive
```

```
[ ] network.compile(optimizer='sgd',
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])
```

Preparing the image data

```
[ ] train_images = train_images.reshape((60000, 28 * 28))
    train_images = train_images.astype('float32') / 255.
    test_images = test_images.reshape((10000, 28 * 28))
    test_images = test_images.astype('float32') / 255.
```

Preparing the labels

```
[ ] from keras.utils import to_categorical
    train_labels = to_categorical(train_labels)
```

ANN lab work.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 28

+ Code + Text

```
313/313 [==============================] - 1s 2ms/step - loss: 0.5788 - accuracy: 0.8627
Test Accuracy: 0.86270
0.8626999855041504
```

```python
from keras.utils import plot_model
plot_model(network, to_file='model.png')
import matplotlib.pyplot as plt
history =network.fit(train_images, train_labels, validation_split=0.33,epochs=5, batch_size=512)
history_dict = history.history
print(history_dict.keys())
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()


# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
Epoch 1/5
79/79 [==============================] - 1s 13ms/step - loss: 0.5990 - accuracy: 0.8539 - val_loss: 0.5805 - val_accuracy: 0.8626
Epoch 2/5
79/79 [==============================] - 1s 12ms/step - loss: 0.5971 - accuracy: 0.8541 - val_loss: 0.5787 - val_accuracy: 0.8629
Epoch 3/5
79/79 [==============================] - 1s 11ms/step - loss: 0.5953 - accuracy: 0.8544 - val_loss: 0.5770 - val_accuracy: 0.8631
Epoch 4/5
```

ANN lab work.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 28

+ Code + Text

```
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

Training and Testing

```python
network.fit(train_images, train_labels, epochs=160, batch_size=512)
```

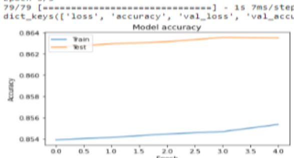
```
Epoch 1/160
118/118 [==============================] - 1s 5ms/step - loss: 2.3521 - accuracy: 0.0993
Epoch 2/160
118/118 [==============================] - 1s 5ms/step - loss: 2.3229 - accuracy: 0.0993
Epoch 3/160
118/118 [==============================] - 1s 5ms/step - loss: 2.3063 - accuracy: 0.0998
Epoch 4/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2954 - accuracy: 0.1135
Epoch 5/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2873 - accuracy: 0.1703
Epoch 6/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2805 - accuracy: 0.2157
Epoch 7/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2741 - accuracy: 0.2738
Epoch 8/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2680 - accuracy: 0.2830
Epoch 9/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2617 - accuracy: 0.2776
Epoch 10/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2552 - accuracy: 0.2899
Epoch 11/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2485 - accuracy: 0.2990
Epoch 12/160
118/118 [==============================] - 1s 5ms/step - loss: 2.2413 - accuracy: 0.3144
Epoch 13/160
118/118 [==============================] - 1s 8ms/step - loss: 2.2337 - accuracy: 0.3244
Epoch 14/160
118/118 [==============================] - 1s 8ms/step - loss: 2.2257 - accuracy: 0.3393
Epoch 15/160
118/118 [==============================] - 1s 8ms/step - loss: 2.2171 - accuracy: 0.3584
```

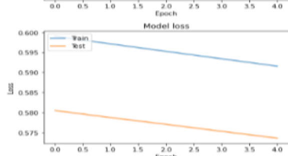
ANN lab work.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 28

+ Code + Text

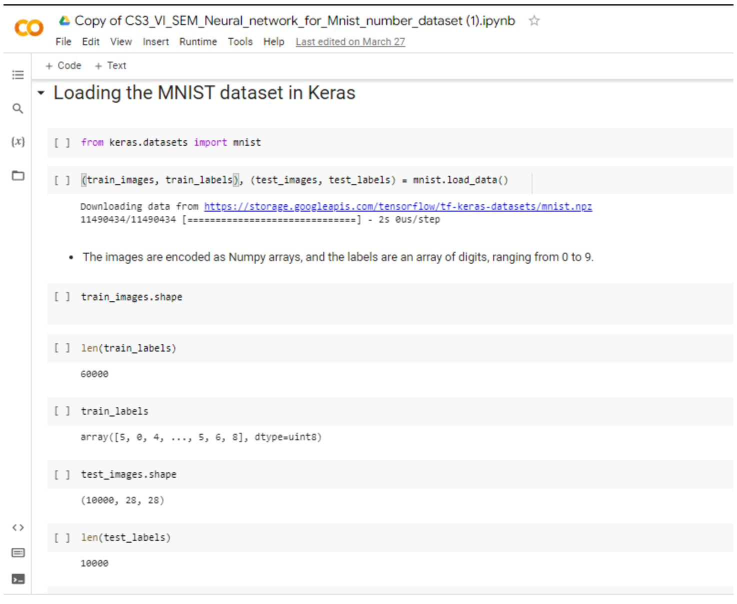
```
79/79 [==============================] - 1s 12ms/step - loss: 0.5971 - accuracy: 0.8541 - val_loss: 0.5787 - val_accuracy: 0.8629
Epoch 3/5
79/79 [==============================] - 1s 11ms/step - loss: 0.5953 - accuracy: 0.8544 - val_loss: 0.5770 - val_accuracy: 0.8631
Epoch 4/5
79/79 [==============================] - 1s 7ms/step - loss: 0.5934 - accuracy: 0.8547 - val_loss: 0.5753 - val_accuracy: 0.8635
Epoch 5/5
79/79 [==============================] - 1s 7ms/step - loss: 0.5916 - accuracy: 0.8553 - val_loss: 0.5736 - val_accuracy: 0.8635
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





EXPERIMENT 8

AIM : Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using CIFAR 100 Multiclass classification data sets with use of use of batch normalization, early stopping and drop out

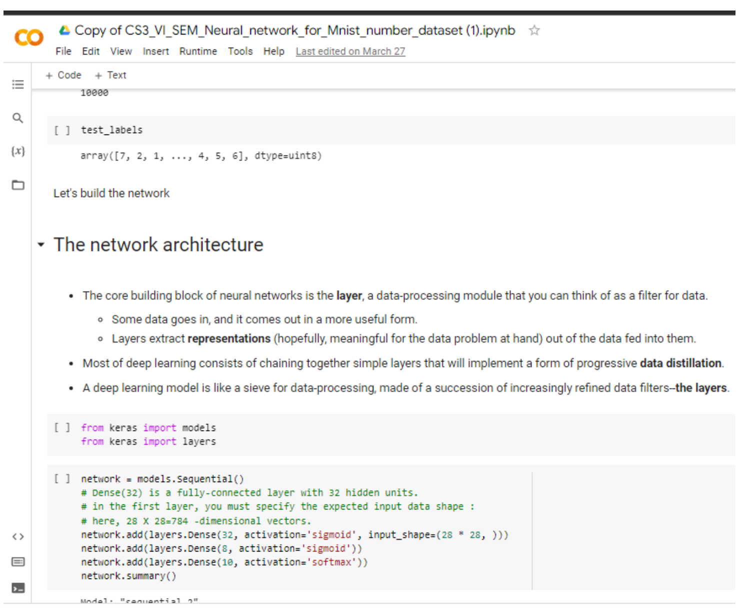

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb
File Edit View Insert Runtime Tools Help Last edited on March 27
+ Code + Text

Loading the MNIST dataset in Keras

```
[ ] from keras.datasets import mnist

[ ] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
    11490434/11490434 [==============================] - 2s 0us/step
```

- The images are encoded as Numpy arrays, and the labels are an array of digits, ranging from 0 to 9.

```
[ ] train_images.shape

[ ] len(train_labels)

    60000

[ ] train_labels

    array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

[ ] test_images.shape

    (10000, 28, 28)

[ ] len(test_labels)

    10000
```


Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb
File Edit View Insert Runtime Tools Help Last edited on March 27
+ Code + Text

```
    10000

[ ] test_labels

    array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

Let's build the network

The network architecture

- The core building block of neural networks is the **layer**, a data-processing module that you can think of as a filter for data.
  - Some data goes in, and it comes out in a more useful form.
  - Layers extract **representations** (hopefully, meaningful for the data problem at hand) out of the data fed into them.
- Most of deep learning consists of chaining together simple layers that will implement a form of progressive **data distillation**.
- A deep learning model is like a sieve for data-processing, made of a succession of increasingly refined data filters--**the layers**.

```
[ ] from keras import models
    from keras import layers

[ ] network = models.Sequential()
    # Dense(32) is a fully-connected layer with 32 hidden units.
    # in the first layer, you must specify the expected input data shape :
    # here, 28 X 28=784 -dimensional vectors.
    network.add(layers.Dense(32, activation='sigmoid', input_shape=(28 * 28, )))
    network.add(layers.Dense(8, activation='sigmoid'))
    network.add(layers.Dense(10, activation='softmax'))
    network.summary()

    Model: "sequential_2"
```

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
network = models.Sequential()
# Dense(32) is a fully-connected layer with 32 hidden units.
# in the first layer, you must specify the expected input data shape :
# here, 28 X 28=784 -dimensional vectors.
network.add(layers.Dense(32, activation='sigmoid', input_shape=(28 * 28, )))
network.add(layers.Dense(8, activation='sigmoid'))
network.add(layers.Dense(10, activation='softmax'))
network.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 32)                25120

 dense_7 (Dense)             (None, 8)                 264

 dense_8 (Dense)             (None, 10)                90

=================================================================
Total params: 25,474
Trainable params: 25,474
Non-trainable params: 0
_________________________________________________________________
```

- Our network consists of a sequence of two *Dense* layers, which are densely connected (also called *fully connected*) neural layers.
- The second (and last) layer is a **10-way** *softmax* layer, which means it will return an array of **10** probability scores. Each score will be the probability that the current digit image belongs to one of our 10 digit classes.

The compilation step

- To make the network ready for training, we need to pick three more things, as part of the **compilation** step:
  - **A loss function:** How the network will be able to measure its performance on the training data, and thus how it will be able to steer

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

Preparing the image data

Before training, we will preprocess the data by reshaping it into the shape the network expects and scaling it so that all values are in the $[0-1]$ interval.

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255.
```

```
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255.
```

Preparing the labels

We also need to categorically encode the labels.

```
from keras.utils import to_categorical
```

```
train_labels = to_categorical(train_labels)
train_labels
```

```
array([[[[[0., 1.],
          [1., 0.]],

         [[0., 1.],
          [1., 0.]],

         [[0., 1.],
          [1., 0.]],
```

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
        [[0., 1.],
         [1., 0.]],

        [[0., 1.],
         [1., 0.]],

        ...,

        [[0., 1.],
         [1., 0.]],

        [[1., 0.],
         [0., 1.]],

        [[0., 1.],
         [1., 0.]]]], dtype=float32)
```

```
[ ] test_labels = to_categorical(test_labels)
    test_labels
```

```
array([[[[0., 1.],
         [1., 0.]],

        [[0., 1.],
         [1., 0.]],

        [[0., 1.],
         [1., 0.]],

        ...,

        [[1., 0.],
         [0., 1.]],

        [[0., 1.],
         [1., 0.]],

        [[0., 1.],
```

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

Training and Testing

We are now ready to train the network, which in Keras is done via a call to the network's fit method--we fit the model to its training data:

```
[ ] network.fit(train_images, train_labels, epochs=500, batch_size=512)
```

- Two quantities are displayed during training:
  - The loss of the network over the training data
  - The accuracy of the network over the training data
- We quickly reach an accuracy of $0.9886 (98.86\%)$ on the training data.

- Now let's check that the model performs well on the test set, too:

```
[ ] test_loss, test_acc = network.evaluate(test_images, test_labels)

313/313 [==============================] - 1s 2ms/step - loss: 0.2511 - accuracy: 0.9317
```

```
[ ] print('Test Accuracy: {:.5f} '.format(test_acc))

Test Accuracy: 0.93170
```

```
[ ] test_acc

0.9316999912261963
```

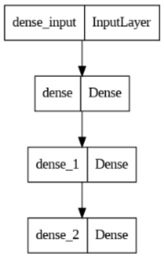
- The test-set accuracy turns out to be 97.780%--that is quite a bit lower than the training set accuracy. This gap between training and test accuracy is an example of **overfitting**:the fact that the ML models tend to perform worse on new data than on their training data.

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
[ ] from keras.utils import plot_model
    plot_model(network, to_file='model.png')
```



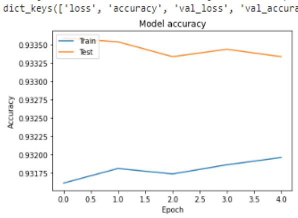

```
import matplotlib.pyplot as plt
history =network.fit(train_images, train_labels, validation_split=0.33,epochs=5, batch_size=512)
history_dict = history.history
print(history_dict.keys())
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

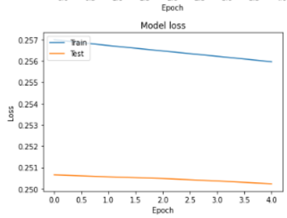

Copy of CS3_VI_SEM_Neural_network_for_Mnist_number_dataset (1).ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

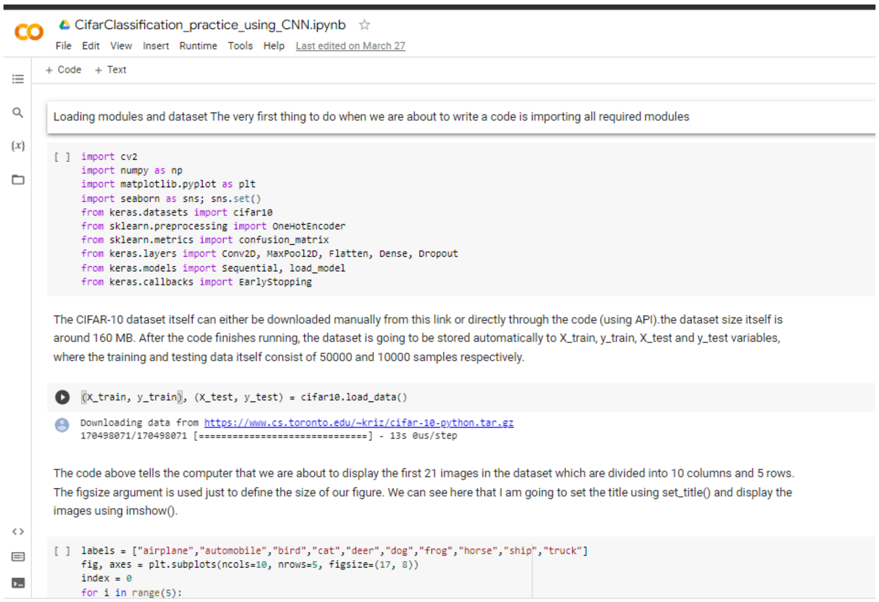

```
Epoch 3/5
79/79 [==============================] - 1s 7ms/step - loss: 0.2565 - accuracy: 0.9317 - val_loss: 0.2505 - val_accuracy: 0.9333
Epoch 4/5
79/79 [==============================] - 1s 7ms/step - loss: 0.2562 - accuracy: 0.9319 - val_loss: 0.2504 - val_accuracy: 0.9334
Epoch 5/5
79/79 [==============================] - 1s 7ms/step - loss: 0.2560 - accuracy: 0.9320 - val_loss: 0.2502 - val_accuracy: 0.9333
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





EXPERIMENT 9

AIM: Build an Convolutional Neural Network by implementing the Backpropagation algorithm and test the same using CIFAR 100 Multiclass classification data sets.



CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

Loading modules and dataset The very first thing to do when we are about to write a code is importing all required modules

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from keras.datasets import cifar10
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout
from keras.models import Sequential, load_model
from keras.callbacks import EarlyStopping
```

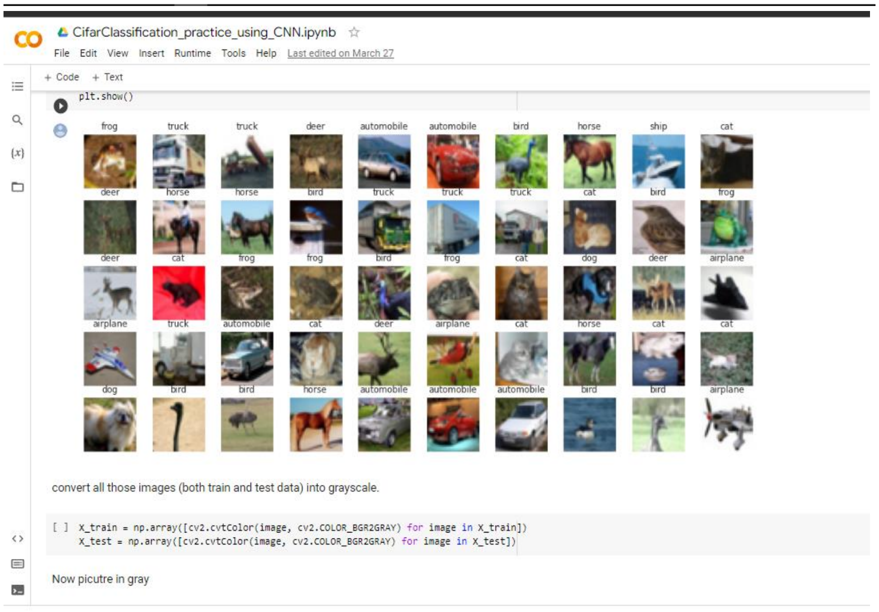
The CIFAR-10 dataset itself can either be downloaded manually from this link or directly through the code (using API).the dataset size itself is around 160 MB. After the code finishes running, the dataset is going to be stored automatically to X_train, y_train, X_test and y_test variables, where the training and testing data itself consist of 50000 and 10000 samples respectively.

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
```

The code above tells the computer that we are about to display the first 21 images in the dataset which are divided into 10 columns and 5 rows. The figsize argument is used just to define the size of our figure. We can see here that I am going to set the title using set_title() and display the images using imshow().

```
labels = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
fig, axes = plt.subplots(ncols=10, nrows=5, figsize=(17, 8))
index = 0
for i in range(5):
```



CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
plt.show()
```


convert all those images (both train and test data) into grayscale.

```
X_train = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in X_train])
X_test = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in X_test])
```

Now picutre in gray

CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

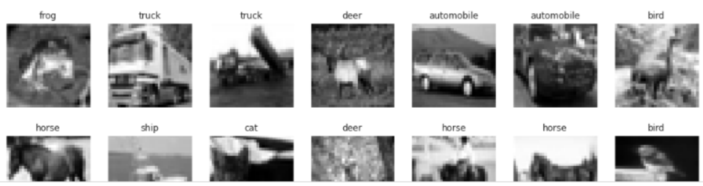
convert all those images (both train and test data) into grayscale.

```
X_train = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in X_train])
X_test = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in X_test])
```

Now picutre in gray

Double-click (or enter) to edit

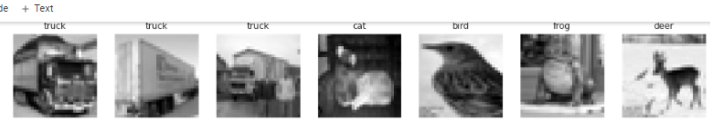
```
fig, axes = plt.subplots(ncols=7, nrows=3, figsize=(17, 8))
index = 0
for i in range(3):
    for j in range(7):
        axes[i,j].set_title(labels[y_train[index][0]])
        axes[i,j].imshow(X_train[index], cmap='gray')
        axes[i,j].get_xaxis().set_visible(False)
        axes[i,j].get_yaxis().set_visible(False)
        index += 1
plt.show()
```



CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text



Double-click (or enter) to edit

normalize array values. We know that by default the brightness of each pixel in any image are represented using a value which ranges between 0 and 255. In order for neural network to work best, we need to convert this value such that it's going to be in the range between 0 and 1.

```
X_train  = X_train/255
X_test  = X_test/255
```

Data preprcessing

```
one_hot_encoder = OneHotEncoder(sparse_output=False)
one_hot_encoder.fit(y_train)
```

```
OneHotEncoder
OneHotEncoder(sparse_output=False)
```

```
one_hot_encoder = OneHotEncoder( sparse_output=False)
one_hot_encoder.fit(y_test)
```

```
OneHotEncoder
OneHotEncoder(sparse_output=False)
```

CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
[ ]
OneHotEncoder
OneHotEncoder(sparse_output=False)
```

```
[ ] y_train = one_hot_encoder.transform(y_train)
    y_test = one_hot_encoder.transform(y_test)
```

the shape of X_train and X_test, the size will be (50000, 32, 32) and (10000, 32, 32) respectively. Well, actually this shape is not acceptable by Conv2D layer that we are going to implement. So, we need to reshape those two arrays using the following code:

```
[ ] X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
    X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

Now our X_train and X_test shapes are going to be (50000, 32, 32, 1) and (10000, 32, 32, 1), where the number 1 in the last position indicates that we are now using only 1 color channel (gray).

```
[ ] input_shape = (X_train.shape[1], X_train.shape[2], 1)
```

Double-click (or enter) to edit

```
[ ] model = Sequential()
    model.add(Conv2D(16, (3, 3), activation='relu', strides=(1, 1),
        padding='same', input_shape=input_shape))
    model.add(Conv2D(32, (3, 3), activation='relu', strides=(1, 1),
        padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', strides=(1, 1),
        padding='same'))
    model.add(MaxPool2D((2, 2)))
    model.add(Conv2D(32, (3, 3), activation='relu', strides=(1, 1)
```

CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
[ ] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 16)        160

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        4640

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_3 (Conv2D)           (None, 16, 16, 32)        18464

 conv2d_4 (Conv2D)           (None, 16, 16, 32)        9248

 conv2d_5 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 256)               1048832

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 64)                4160

 dense_4 (Dense)             (None, 10)                650

=================================================================
```

CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Last edited on March 27

+ Code + Text

```
[ ] print(X_train.shape)

(50000, 32, 32, 1)
```

```
[ ] history = model.fit(X_train, y_train, epochs=30
                        , batch_size=8, validation_data=(X_test, y_test))

Epoch 1/30
6250/6250 [==============================] - 59s 7ms/step - loss: 2.3032 - acc: 0.0978 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 2/30
6250/6250 [==============================] - 48s 8ms/step - loss: 2.3029 - acc: 0.0980 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 3/30
1088/6250 [====>.........................] - ETA: 35s - loss: 2.3029 - acc: 0.0975
```

```
# Generate generalization metrics
score = model.evaluate(X_test, y_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_acc'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

```
[ ] plt.plot(history.history["acc"])
```

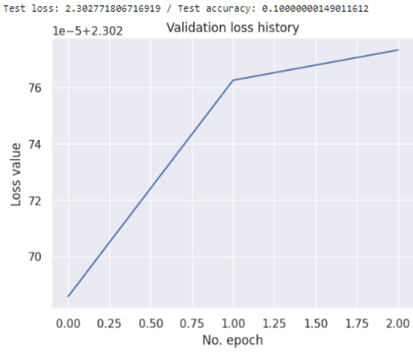
CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Save failed

+ Code + Text

```
# Plot history: Accuracy
plt.plot(history.history['val_acc'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

```
Test loss: 2.302771806716919 / Test accuracy: 0.10000000149011612
```



Validation accuracy history

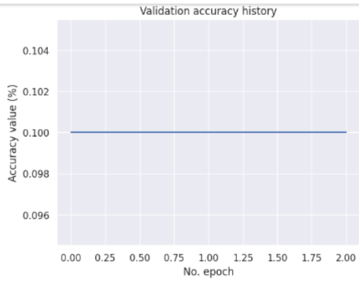
Automatic saving failed. This file was updated remotely or in another tab. Show diff

✓ 0s completed at 12:28 AM

CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Save failed

+ Code + Text




```
plt.plot(history.history["acc"])
plt.plot(history.history["val_acc"])
plt.show()
```

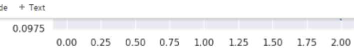
Automatic saving failed. This file was updated remotely or in another tab. Show diff

0s completed at 12:28 AM

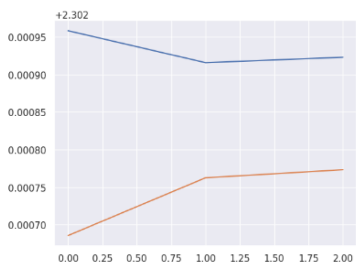
CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Save failed

+ Code + Text




```
[25] plt.plot(history.history["loss"])
     plt.plot(history.history["val_loss"])
     plt.show()
```




```
predictions = model.predict(X_test)
predictions = one_hot_encoder.inverse_transform(predictions)
```

```
313/313 [==============================] - 1s 3ms/step
```

```
[27] y_test = one_hot_encoder.inverse_transform(y_test)
```

```
[28] cm = confusion_matrix(y_test, predictions)
```


Automatic saving failed. This file was updated remotely or in another tab. Show diff

0s completed at 12:28 AM

CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Save failed

+ Code + Text

```python
plt.figure(figsize=(9,9))
sns.heatmap(cm, cbar=False, xticklabels=labels, yticklabels=labels, fmt="d", annot=True, cmap=plt.cm.Blues)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Actual

airplane	0	0	0	0	0	0	1000	0	0	0
automobile	0	0	0	0	0	0	1000	0	0	0
bird	0	0	0	0	0	0	1000	0	0	0
cat	0	0	0	0	0	0	1000	0	0	0
deer	0	0	0	0	0	0	1000	0	0	0
dog	0	0	0	0	0	0	1000	0	0	0
frog	0	0	0	0	0	0	1000	0	0	0
horse	0	0	0	0	0	0	1000	0	0	0
ship	0	0	0	0	0	0	1000	0	0	0

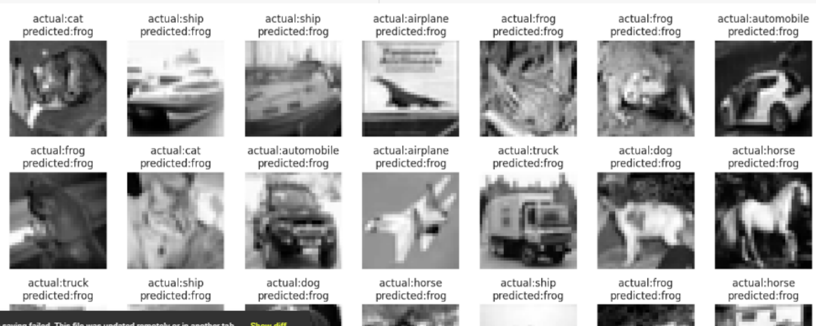
CifarClassification_practice_using_CNN.ipynb

File Edit View Insert Runtime Tools Help Save failed

+ Code + Text

```python
[31] y_test = y_test.astype(int)
predictions = predictions.astype(int)
```

```python
fig, axes = plt.subplots(ncols=7, nrows=3, sharex=False,
    sharey=True, figsize=(17, 8))
index = 0
for i in range(3):
    for j in range(7):
        axes[i,j].set_title('actual:' + labels[y_test[index][0]] + '\n'
                            + 'predicted:' + labels[predictions[index][0]])
        axes[i,j].imshow(X_test[index], cmap='gray')
        axes[i,j].get_xaxis().set_visible(False)
        axes[i,j].get_yaxis().set_visible(False)
        index += 1
plt.show()
```



Automatic saving failed. This file was updated remotely or in another tab. Show diff

✓ 0s completed at 12:28 AM

EXPERIMENT 10

AIM: Study of Transfer Learning (VGG 16)

Transfer learning is a machine learning technique where a pre-trained model is used as a starting point for a new task. In the context of computer vision, a pre-trained model trained on a large dataset such as ImageNet can be used as a starting point for other computer vision tasks such as object detection, classification, and segmentation.

One popular pre-trained model used in transfer learning is VGG16. VGG16 is a deep convolutional neural network model that was developed by the Visual Geometry Group at the University of Oxford. It was trained on the ImageNet dataset, which contains over 14 million images and 1000 object categories.

VGG16 consists of 16 convolutional layers followed by 3 fully connected layers. The convolutional layers are arranged in blocks, with each block containing multiple convolutional layers followed by a max pooling layer. The fully connected layers at the end of the network perform classification on the output of the convolutional layers.

To use VGG16 for transfer learning, the fully connected layers at the end of the network are removed, and the output of the last convolutional layer is used as input to a new network. The new network can be trained on a smaller dataset for a different computer vision task, such as object detection or segmentation.

The pre-trained VGG16 model is available in popular deep learning frameworks such as TensorFlow and Keras. Here's an example of using VGG16 for image classification in Keras:

```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

# Load the pre-trained VGG16 model
model = VGG16(weights='imagenet')

# Load an image and preprocess it for VGG16
img_path = 'cat.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Use the pre-trained model to predict the class of the image
preds = model.predict(x)
```

```
print('Predicted:', decode_predictions(preds, top=3)[0])
```

This code loads the pre-trained VGG16 model, loads an image, preprocesses it for VGG16, and uses the model to predict the class of the image. The output of the model is a probability distribution over 1000 ImageNet categories, and **decode_predictions()** is used to convert the probabilities to human-readable class labels.

EXPERIMENT 11

AIM : Implementation of RNN

RNN.ipynb

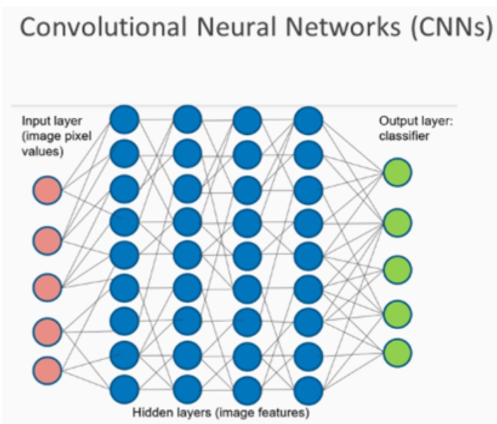
File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

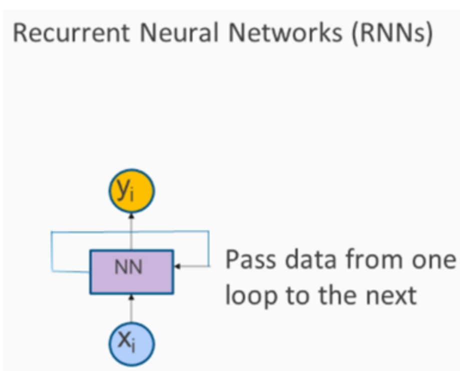
Recurrent neural networks(RNN) with long short-term memory(LSTM)

Recurrent means the output at the current time step becomes the input to the next time step. At each element of the sequence, the model considers not just the current input, but what it remembers about the preceding elements.

As powerful as convolutional neural networks (CNNs) are, they don't handle sequential data so well; however, they are great for non-sequential tasks, such as image classification.

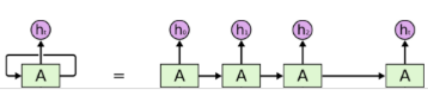


Recurrent neural networks (RNNs), which really are state of the art, can handle sequential tasks. An RNN consists of CNNs where data is received in a sequence.



Data coming in a sequence (xi) goes through the neural network and we get an output (yi). The output is then fed through to another iteration and forms a loop. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.

Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition, language translation, video identification, and text generation.



Advantages of Recurrent Neural Network

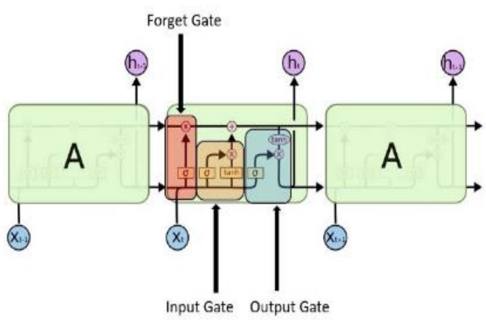
1. RNN can model sequence of data so that each sample can be assumed to be dependent on previous ones
2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighbourhood.

Disadvantages of Recurrent Neural Network

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

What is Long Short Term Memory (LSTM)?

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:



RNN.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive

Input gate — discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.

Forget gate — discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state (ht-1) and the content input(Xt) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state Ct−1

Output gate — the input and the memory of the block is used to decide the output. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1 and multiplied with output of Sigmoid.

Details

```
#import required packages/library
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM#, CuDNNLSTM
```

Similar to before, we load in our data, and we can see the shape again of the dataset and individual samples:

```
mnist = tf.keras.datasets.mnist  # mnist is a dataset of 28x28 images of handwritten digits and their labels
(x_train, y_train),(x_test, y_test) = mnist.load_data()  # unpacks images to x_train/x_test and labels to y_train/y_test

x_train = x_train/255
x_test = x_test/255

print("x_train =",x_train.shape)
print("y_train =",y_train.shape)
print("x_test =",x_test.shape)
print("y_test =",y_test.shape)

print(x_train[0].shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
x_train = (60000, 28, 28)
y_train = (60000,)
x_test = (10000, 28, 28)
y_test = (10000,)
(28, 28)
```

Recall we had to flatten this data for the regular deep neural network. In this model, we're passing the rows of the image as the sequences. So basically, we're showing the the model each pixel row of the image, in order, and having it make the prediction. (28 sequences of 28 elements)

RNN.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

Recall we had to flatten this data for the regular deep neural network. In this model, we're passing the rows of the image as the sequences. So basically, we're showing the the model each pixel row of the image, in order, and having it make the prediction. (28 sequences of 28 elements)

```
[ ] model = Sequential()

    # IF you are running with a GPU, try out the CuDNNLSTM layer type instead (don't pass an activation, tanh is required)
    model.add(LSTM(128, input_shape=(x_train.shape[1:]), activation='relu', return_sequences=True))
    model.add(Dropout(0.2))

    #return_sequences= True   # This flag is used for when you're continuing on to another recurrent layer.

    model.add(LSTM(128, activation='relu'))
    model.add(Dropout(0.1))

    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(10, activation='softmax')) #output layer
```

This should all be straight forward, where rather than Dense or Conv, we're just using LSTM as the layer type. The only new thing is return_sequences. This flag is used for when you're continuing on to another recurrent layer. If you are, then you want to return sequences. If you're not going to another recurrent-type of layer, then you don't set this to true.

```
[ ] model.summary()

    Model: "sequential_1"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     lstm_2 (LSTM)               (None, 28, 128)           80384

     dropout_3 (Dropout)         (None, 28, 128)           0

     lstm_3 (LSTM)               (None, 128)               131584

     dropout_4 (Dropout)         (None, 128)               0

     dense_2 (Dense)             (None, 32)                4128

     dropout_5 (Dropout)         (None, 32)                0

     dense_3 (Dense)             (None, 10)                330

    =================================================================
```

RNN.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

Compile the model

```
opt = tf.keras.optimizers.legacy.Adam(lr=0.001, decay=1e-6)

# Compile model
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=opt,
    metrics=['accuracy'],
)

/usr/local/lib/python3.9/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

Training to the model

```
model.fit(x_train,y_train,
          epochs=3,verbose=1)

Epoch 1/3
1875/1875 [==============================] - 180s 95ms/step - loss: 0.6331 - accuracy: 0.7919
Epoch 2/3
1875/1875 [==============================] - 172s 92ms/step - loss: 0.1511 - accuracy: 0.9597
Epoch 3/3
1875/1875 [==============================] - 162s 86ms/step - loss: 0.1031 - accuracy: 0.9732
<keras.callbacks.History at 0x7f80cc1ee370>
```

```
[ ] score = model.evaluate(x_test, y_test)
    print('Test score:', score[0])
    print('Test accuracy:', score[1])

313/313 [==============================] - 10s 31ms/step - loss: 0.0600 - accuracy: 0.9818
Test score: 0.05999040976166725
Test accuracy: 0.9818000197410583
```